Cisco Webex Android SDK Transition Guide

> The Cisco Webex™ Android SDK Version 3.0.0.2-beta

The Cisco Webex Android SDK Version 3.0.0.2-beta is built in kotlin language. So, if you have developed an app with Cisco Webex SDK version 2.X.X, this guide will help you in transitioning from Webex SDK version 2 to version 3.

This SDK is built with **Android SDK Tools 29** and requires **Android API Level 24** or later.

Table of Contents

Install

Assuming you already have an Android project, e.g. *KitchenSinkApp*, for your Android app, here are the steps to integrate the Cisco Webex Android SDK into your project using Gradle:

1. Put AAR file in libs folder of your Android project
2. Open the project level `build.gradle` file and add the following lines under the repositories tag, which is nested under allprojects.

```
allprojects {
  repositories {
      jcenter()
      google()
      flatDir { dirs 'aars'}
  }
}
```

3. Add the following dependency in module level Gradle file and press sync-now assuming AAR file name is "Webex-SDK-v3.0.0.2-beta.aar". You should replace this name accordingly.

```
implementation files('libs/Webex-SDK-v3.0.0.2-beta.aar')
```

Note

Since, this SDK is built in Kotlin language. So, the transition between java and kotlin has to be taken care if app is built in Java language.

Examples of transition

Here are some examples in Java to transition from Webex SDK v2 to Webex SDK v3:

1. If you're using JWTAuthenticator authorization. In this version, we have added completion handler which gets invoked when authorization process is done.

   Previous syntax:

   ```
   jwtAuthenticator.authorize(jwt);
   ```

   Current syntax using CompletionHandler:

   ```
   jwtAuthenticator.authorize(jwt, new CompletionHandler<Void>() {
    @Override
        public void onComplete(Result<Void> result) {
            //Authorization process is completed
        }
    });
   ```

2. Instantiating `OAuthWebViewAuthenticator`. We were using client scope as a constructor argument which is now removed because scope is added internally.

   Previous syntax:

   ```
   OAuthWebViewAuthenticator authenticator = new
   OAuthWebViewAuthenticator(clientId, clientSec, scope, redirect);
   ```

   Current syntax:

   ```
   OAuthWebViewAuthenticator authenticator = new
   OAuthWebViewAuthenticator(clientId, clientSec, redirect);
   ```

3. `oAuth2.isAuthorized()` is a work in progress, you should use `webex.attemptToLoginWithCachedUser()` instead

   Previous syntax:
   `oAuth2.isAuthorized()` is used to check if user is already authorized.

   ```
    if (oAuth2.isAuthorized()) {
        //User is already authorized
    }
   ```

Current syntax:
`webex.attemptToLoginWithCachedUser(CompletionHandler handler)`

This method is used to login with already cached data. It lets you in if you are already authenticated. This method takes in completion handler as a parameter which returns `Result.isSuccessful()` as `true` if user is authorised.
Else, user is not authorized, needs to login again.

Usage

```
webex.attemptToLoginWithCachedUser(result -> {
    if(result.isSuccessful()) {
        //user is authorised
    } else {
        //user is not authorized, needs to login again.
    }
});
```

4. `Phone.register()` API is no more required because this happens at the time of login internally. You have to comment/remove this.

5. `Phone.deregister()` API is no more required. You have to comment/remove this.

6. In case of dial API, isModerator and pin argument of MediaOption are in progress.

   Previous syntax:

```
public void dial(String callee, View localView, View remoteView, View
screenSharing, boolean isModerator, String pin) {
         phone.dial(callee, getMediaOption(localView, remoteView,
screenSharing, isModerator, pin), (Result<Call> result) -> {
             if (result.isSuccessful()) {
                 // ...
             }
         });
  }
```

   Current syntax:

```
public void dial(String callee, View localView, View remoteView, View
screenSharing) {
     phone.dial(callee, getMediaOption(localView, remoteView,
screenSharing), (Result<Call> result) -> {
         if (result.isSuccessful()) {
             // ...
         }
     });
 }
```

7. Changes done in #6 also needs to be followed for `Phone.answer()` API

In Progress APIs

Below APIs are a work in progress which will be available in subsequent releases

1. letIn APIs.
2. AuxStream APIs.
3. MediaOption Moderator APIs.
4. MediaOption Pin APIs.
5. Authenticator `getToken`, `deauthorize`, `authorize`, `refreshToken` APIs.